

DTIC FILE COPY

RADC-TR-90-90
Final Technical Report
May 1990

AD-A224 490



A SURVEY OF OBJECT-ORIENTED DATABASE TECHNOLOGY

George Mason University

Roshan Thomas, Akhil Agrawal, Sushil Jajodia, Boris Kogan

DTIC
ELECTE
JUL 06 1990
S D Cy D

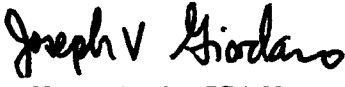
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

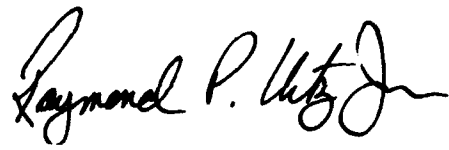
Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 07 5 003

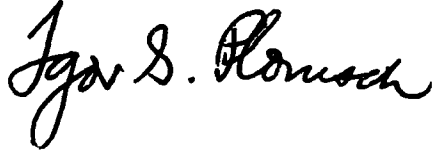
This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-90 has been reviewed and is approved for publication.

APPROVED: 
JOSEPH V. GIORDANO
Project Engineer

APPROVED: 
RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:


IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

| REPORT DOCUMENTATION PAGE | | | Form Approved OPM No. 0704-0188 | |
|---|---|--|--|--|
| <small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of Management and Budget, Washington, DC 20503.</small> | | | | |
| 1. AGENCY USE ONLY (Leave Blank) | | 2. REPORT DATE May 1990 | | 3. REPORT TYPE AND DATES COVERED Final Jun 89 to Dec 89 |
| 4. TITLE AND SUBTITLE A SURVEY OF OBJECT-ORIENTED DATABASE TECHNOLOGY | | | 5. FUNDING NUMBERS C - F30602-88-D-0028 PE - 35167G PR - 1068 TA - 01 WU - P4 | |
| 6. AUTHOR(S) Roshan Thomas, Akhil Agrawal, Sushil Jajodia, Boris Kogan | | | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) George Mason University Dept of Information Systems & Systems Engineering 4400 University Drive Fairfax VA 22030-4444 | | | 8. PERFORMING ORGANIZATION REPORT NUMBER | |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Air Development Center (COTD) Griffiss AFB NY 13441-5700 | | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER RADC-TR-90-90 | |
| 11. SUPPLEMENTARY NOTES RADC Project Engineer: Joseph V. Giordano/COTD/(315) 330-2925 | | | | |
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited. | | | 12b. DISTRIBUTION CODE | |
| 13. ABSTRACT (Maximum 200 words) For many applications, traditional data base models and technologies have proven inadequate, and this has led to research and development of database technologies based on the object-oriented paradigm. In this report, some of the recent research and development in object-oriented databases are surveyed. The core concepts that are being considered in developing a unified object-oriented data model are discussed, followed by some of the architectural issues involved in developing object-oriented database management systems. Technical summaries of two prototype object-oriented database products, GEMSTONE AND ORION, are given. Also, some of the exciting future research directions are highlighted. | | | | |
| 14. SUBJECT TERMS Object-oriented, data base management systems, trusted systems, multilevel security | | | 15. NUMBER OF PAGES 52 | |
| | | | 16. PRICE CODE | |
| 17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED | 18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED | 19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED | 20. LIMITATION OF ABSTRACT UL | |

ACKNOWLEDGEMENTS

This work was partially supported by the U. S. Air Force, Rome Air Development Center through subcontract # RI-64155X of prime contract # F30602-88-D-0028, Task B-9-3622 with University of Dayton. We are indebted to RADC for making this work possible.

| | |
|--------------------|--|
| Accession For | |
| NTIS CRA&I | <input checked="checked" type="checkbox"/> |
| DTIC TAB | <input type="checkbox"/> |
| Unannounced | <input type="checkbox"/> |
| Justification | |
| By | |
| Distribution / | |
| Availability Codes | |
| Dist | Avail and/or Special |
| A-1 | |



Contents

| | | |
|----------|---|-----------|
| 1 | INTRODUCTION | 2 |
| 2 | TOWARDS A UNIFIED CORE MODEL | 5 |
| 2.1 | Data Model Issues | 6 |
| 3 | ARCHITECTURAL ISSUES | 9 |
| 3.1 | Schema Evolution | 9 |
| 3.1.1 | Solution for Instances | 10 |
| 3.1.2 | Solution for Users of Instances | 11 |
| 3.2 | Concurrency Control | 12 |
| 3.3 | Version Control and Management | 14 |
| 3.4 | Query Processing and Indexing | 16 |
| 3.4.1 | Query Evaluation Strategies | 16 |
| 3.4.2 | Indexing Techniques | 17 |
| 4 | COMMERCIAL AND PROTOTYPE SYSTEMS | 18 |
| 4.1 | The ORION Database System | 19 |
| 4.1.1 | Data Model Support | 20 |
| 4.1.2 | The ORION Architecture | 21 |
| 4.1.3 | Multimedia Information Support | 24 |
| 4.2 | The GEMSTONE Database System | 24 |
| 4.2.1 | Data Model Support | 24 |
| 4.2.2 | The GEMSTONE Architecture | 25 |
| 4.2.3 | Language Support | 27 |
| 5 | RESEARCH DIRECTIONS | 28 |
| 5.1 | Integrating Object-Oriented Languages and Databases | 28 |
| 5.2 | Transaction Management | 29 |
| 5.3 | Distributed Object-Oriented Database Systems | 30 |
| 5.4 | Active Database Systems | 32 |
| 5.5 | Tools and Development Environments | 32 |
| 5.6 | Optimization Techniques | 33 |

A SURVEY OF OBJECT-ORIENTED DATABASE TECHNOLOGY

Roshan Thomas

Akhil Agrawal

Sushil Jajodia

Boris Kogan

*Department of Information Systems and Systems Engineering
George Mason University, Fairfax, VA 22030*

Abstract

For many applications, traditional data base models and technologies have proved inadequate and this has lead to research and developments of database technologies based on the object-oriented paradigm. In this paper, we survey some of the recent research and developments in object-oriented databases. We discuss the core concepts that are being considered in developing a unified object-oriented data model followed by some of the architectural issues involved in developing object-oriented database management systems. We also give technical summaries of two prototype object-oriented database products GEMSTONE and ORION and highlight some of the exciting future research directions.

1 INTRODUCTION

It is now widely recognized that conventional database management systems (DBMS) offer poor support and unsatisfactory performance in application areas such as CAD/CAM [3] [7] and engineering, image processing, and office information systems. These applica-

tions require support for the modeling and representation of complex objects and entities whereas conventional database and information management technologies are primarily record-based.

A database is supposed to represent the interesting semantics of an application as completely and accurately as possible. The *data model* incorporated into a database system defines a framework of concepts that can be used to express the miniworld semantics. However, due to the rigid framework of conventional (relational, network, hierarchical) data models there will always be a semantic gap between an application/miniworld and its database representation. Today, object-oriented databases based on the object-oriented data model are an attempt to minimize this semantic gap.

An object-oriented data model is a data model that allows any real-world entity to be modeled exactly as an object. Thus no artificial decomposition into simpler concepts/entities is necessary. Looking in more detail, the object-oriented data model can be seen as having several levels of object-orientation [17]:

- *structural*: if the model allows one to define data structures to represent entities of any complexity.
- *operational*: if the data model includes (generic) operators to deal with complex objects in their entirety.

- *behavioral*: if the data model provides features to define object types of any complexity together with a set of specific operators (abstract data types); instances can then only be used by calling these operators as the internal structure is completely encapsulated to the outside world.

In order to better understand this paper in some perspective, it is worth discussing briefly some of the perceived advantages of object-oriented database management systems:

- Object-oriented databases have the capability to represent and reference objects of complex structures, making them ideal for complex data modeling. This capability is also expected to increase the semantic content of databases.
- Object-oriented databases offer more flexible modeling tools than traditional database systems.
- The design of object-oriented databases incorporate some of the software engineering principles such as data abstraction and information hiding that have proved to be effective in the design of large-scale cost-effective software systems.
- Object-oriented applications can provide version control functions which are required in many data-intensive application domains [9] [14]. This is due to the fact that users in these domains need to generate and experiment with multiple versions of an object before selecting one that satisfies their requirements.

- The object-oriented paradigm will allow protection and security mechanisms to be based on the notion of object which is the natural unit of access control. Also, the paradigm permits description of security requirements in terms familiar to the users as there is a natural correspondence between objects and real-world entities.

2 TOWARDS A UNIFIED CORE MODEL

Today there is a high degree of confusion about what object-oriented means in general and there exists no consensus on the concepts that are to be supported by object-oriented data model. This lack of consensus can be attributed to the fact that object-oriented concepts evolved from three disciplines: programming languages, artificial intelligence, and databases. More recently Beech [8], Kim [20], and Banerjee [4] and other researchers have attempted to present what they believe should constitute the core concepts and data model issues in an object-oriented data model. As our experience and understanding increases, it is only reasonable to expect that many of these concepts will be refined, and many dropped, and others added. At this point, it is worth mentioning the factors and database requirements that need to be considered in the design of a core object-oriented data model. These include:

- requirements such as efficient management of schema evolution, provision of version management, concurrency control, security and protection of objects, transaction

management and recovery techniques;

- the need to provide efficient retrieval and update mechanisms;
- the desirability of increasing the semantic content of databases;
- the desirability that the model provide a basis for describing and designing systems to serve as *information-processing agents*;

2.1 Data Model Issues

We will now highlight some of the core concepts and issues for an object-oriented data model as mentioned in the current literature:

1. **Object.** In object-oriented systems and languages, any real-world entity is uniformly modeled as an object. Every object is associated with a unique identifier. An object is something that can be created, deleted, has an identity and a state, and one that exhibits behavior based on inputs received. The state of an object is the set of values for the attributes of the object. Its behavior is encapsulated in a set of methods(program codes) that manipulate or return the state of the object. An object communicates with other objects by passing messages. For each message accepted by an object, there is a corresponding method executed by the object. It is important to note that only messages and their responses are visible from outside the object; the internal methods and attributes are totally hidden.

The specification of an object consists of an interface part and an implementation part. Once the interface to the object is defined, no operations other than the ones specified in the interface can be performed (this is similar to abstract data types in programming languages). Thus the behavior of an object can be specified, implemented, and understood without reference to the actual implementation of the methods (enforcing the principle of information hiding). In fact, this ability to use an object as an encapsulation mechanism is seen as a major benefit of object-oriented systems.

2. **Class.** Objects that share the same set of attributes and methods are grouped into a class. An object must belong to only one class as an instance of that class. An object is related to its class by the *instance-of* relationship. A class is thus a template (also known as a type in Beech's model) and similar to an abstract data type. A primitive class is one which has associated instances, but which has no attributes, such as integer, string, boolean.
3. **Class Hierarchy.** Object-oriented models allow the user to derive new subclasses from existing classes. A subclass inherits all the attributes and methods of the existing class known as the superclass. It is thus possible to define hierarchies of classes. The concept of a class hierarchy and inheritance of attributes and methods along the hierarchy is what distinguishes object-oriented programming from pro-

gramming with abstract data types. If a class inherits from only one superclass it is seen as *single inheritance*. Inheriting from more than one superclass results in *multiple inheritance* and this can cause naming, typing and other conflicts to occur.

4. **Composite Object.** Many applications require the ability to define and manipulate a set of objects as a single entity. To support this, some object-oriented database products such as ORION provide what is known as *composite objects* [31]. A composite object is defined to be an object with a hierarchy of exclusive component objects. The hierarchy of classes to which these component objects belong is known as a *composite object hierarchy*.

The class hierarchy in an object-oriented data model captures the IS-A relationship between a superclass and its subclasses while a composite object hierarchy captures the IS-PART-OF relationship between a parent class and its component classes. Composite objects add to the integrity features of an object-oriented data model through the notion of dependent objects. A *dependent object* is one whose existence depends on the existence of other objects and that is owned by exactly one object. As such, a dependent object cannot be created if its owner does not already exist. Composite objects also offer an opportunity for performance improvement. For example, in the ORION system a composite object is considered as a unit for clustering related objects on disk. This is because, if an application accesses the

root/parent object, it is often likely to access all (or most) dependent objects as well.

3 ARCHITECTURAL ISSUES

What are the key architectural issues that need to be taken into account in the design of an object-oriented database management system? The object-oriented paradigm requires us to investigate new approaches for handling concurrency control, version control and other issues. The paradigm thus offers new challenges as well as opportunities.

3.1 Schema Evolution

We will now examine the problem of type/schema evolution [5] in an object-oriented database environment. In application environments such as CAD/CAM a user needs the flexibility to change schemas/type definitions. The issue of change poses problems in object-oriented databases due to the persistency and sharing of objects. This is because type changes can result in multiple versions of a type causing incompatibilities. For example, existing application programs which manipulate objects of the type may fail when executed on newly created instances. Moreover, programs written for the new instances may fail on instances of the type created before the change. Skarra and Zdonik [44] [45] have proposed a solution in which changes in a type are hidden such that the objects of the type created before or after the change may be used interchangeably by

programs. In this section we summarize this approach.

Type changes can have two effects:

- **Effect on instances.** A change in a type definition may affect instances of the type and its subtypes (due to inheritance) already in the database. Information stored in the instances may be missing, garbled, or undefined according to the current type definition.
- **Effect on users of instances.** A change in a type definition may affect programs that use objects of the type. Moreover, programs that use objects of its supertypes may also be affected since an object of a type may be used in the same context as that of its supertype.

3.1.1 Solution for Instances

In order for instances in the database to remain meaningful after a type change, either the instances must be *coerced/converted* to the new definition or a new version of the type must be created, leaving the old version intact.

- **Coercion.** The system provides an operation defined by type that converts an instance of one type into that of another. One of the disadvantages of coercion mechanism is that during the reconfiguration of instances, properties and information they contain are discarded if not present in the new type. Values for properties

which are defined in the new type may not be available for instances of the old type. Furthermore, coercion cannot handle the problem of dealing with programs that use objects of the type.

- **Versions.** In this approach, version set is used to capture an ordered collection of all incarnations of a particular type definition. An object is bound throughout its life to a single version of a type; its properties and operations are defined by that one particular version of its type. Coercion may be used in conjunction with version sets as it may be desirable to convert instances of one version into those of another version of the same type.

3.1.2 Solution for Users of Instances

Interobject errors which occur during a program's use of a changed type's object may be detected by either the program or the object. In both cases, the errors occur because every object is strictly bound to a single version of its type. Further, different versions of the same type may define different interfaces for objects. Thus, what we need is an abstraction of the interface defined by a type to which we can bind objects (as opposed to binding to single version of the type). To provide this abstract interface, a *version set interface* which is an inclusive summary of all the type's versions is proposed. Every property and operation ever defined by a version of the type and every value ever declared

valid for properties and operation parameters are represented in the interface. Properties and operations which do not appear in the interface are not valid on any version of the type. Similarly, values which are outside domains defined by the interface are not valid for any version.

3.2 Concurrency Control

Traditional concurrency control mechanisms exploit the semantics of low-level read and write operations in managing concurrent accesses to data while they ignore the semantics of the data items themselves. Object-oriented databases give us the opportunity to exploit the semantics of the objects in designing concurrency control algorithms for managing concurrent accesses and updates [27] [28] [49] [48]. In particular, concurrency control might be influenced by:

- using type-level semantics to achieve greater concurrency
- using type-level semantics to allow nonserializable behavior

Semantic approaches to concurrency control can be broadly divided into two groups:

- **Data Approach:** Here we define concurrency properties on abstract data types according to the semantics of the type and its operations [48]. To give a brief insight, consider the abstract data type `queue(FIFO)`. From the properties of the queue type and the semantics of its operations, we know that the usual "enqueue"

operation which puts an item at the end of the queue and the "dequeue" operation which takes an item from the front of the queue will never conflict as long as there are at least two items in the queue. Thus if we are using semantic concurrency control, we can allow the enqueue and dequeue operations to proceed concurrently. In the data approach, an object offers a uniform, concurrent behavior, regardless of the semantics of the application using it. The approach is modular and allows decentralized concurrency control.

- **Transaction Approach:** Using this approach, we define concurrency properties on transactions according to the semantics of the transactions and the data they manipulate [23],[24], [36]. The transaction approach requires control that is centralized with respect to each group of concurrent transactions, and operation semantics have to be reconsidered in the context of each new transaction.

A second way in which concurrency might differ arises from the fact that in application areas such as those in design environments [3], the nature of transactions may be different from their commercial counterparts. In design environments, we find requirements for cooperative work. Collaborative design argues for a mechanism that relaxes some of the constraints of a strictly serial world. Also, transactions can be very long as a user might for example, keep editing a design document for days. We can thus relax serializability by allowing arbitrary communication between transactions provided we can maintain some

notion of correctness. What this correctness criteria should be is an active area of research.

3.3 Version Control and Management

An *object version* represents an identifiable state of an object. Object versions are either totally ordered as a function of time, or partially ordered in terms of a successor function.

The desirability to provide versions arises at both the system and application levels:

1. To capture the history of an object. In many application such as CAD/CAM several versions may have to exist for a design in order to capture its evolving state.
2. To cope with the problem of changing types, as mentioned in the earlier section.
3. To use object versions for enhancing concurrency control and reliability.

The first use of versions, to capture the history is at the application level, while the third is at the system level. The second, to allow type evolution can be seen at the both the application and system levels.

The issues involved in providing versions and version management functions in object-oriented databases are:

- **Implementing and Preserving Object Identity.** The identity is that property of an object that distinguishes each object from all others [30]. The system must provide a continuous and consistent notion of an object identity across all the versions

of an object. Also, most of the current techniques for implementing object identity (such as physical addresses, or identifier keys) lack either location independence or data (object content) independence. An approach might be using surrogates [15], which are system-generated, globally unique identifiers, completely independent of any physical location.

- **Supporting Linear and Graph Version Structures.** Several parallel versions derived from one initial version could exist at the same time in design applications. Thus the issues here include the complexity of supporting linear and graph version structures as well as operations to create and search these structures.
- **Managing Version States.** When is a version considered safe or stable to be released to others? These and other issues have to be resolved by the provision of different states (with associated permissible operations) for the versions. Common version states in the literature include: transient, working, released, frozen.
- **Storage Management.** How should the different versions of an object be stored? For example, do we have to store each version in its entirety or should we just store the incremental changes between successive versions in a log and reconstruct a version when it is requested, from the log.

3.4 Query Processing and Indexing

In object-oriented databases, the traditional query processing and optimizing strategies (as in relational systems) need to be re-examined as they would have to differ considerably [32] [6]. As an illustration, the scope of a query against a class C in an object-oriented database will be in general the class C and all subclasses of C and their subclasses and so on (as objects may be recursively nested to non-primitive classes).

3.4.1 Query Evaluation Strategies

The nesting of objects implies that to fetch one or more instances of a class, the class and all classes specified as *non-primitive domains* of the attributes of the class must be recursively fetched. This leads to two obvious query evaluation strategies for the tree-like query graph structure:

- **Forward Traversal.** The strategy here is to traverse the classes on the query graph in depth-first order starting from the root of the graph, and following through the successive domains of each complex attribute.
- **Reverse Traversal.** Here we visit the leaf classes first, and then their parents, and so on, working towards the root class.

3.4.2 Indexing Techniques

In object-oriented databases, the class-hierarchy structure and the nested definitions of classes can influence the indexing and storage techniques used to store the objects on stable storage [34]. Recent literature has identified two forms of indexing in the context of object-oriented databases: the class-hierarchy indexing and nested-attribute indexing [33].

Due to the inheritance structure, it may make sense to maintain an index on an attribute for all classes on a class hierarchy rooted at class C, rather than maintaining a separate index on the attribute for each of the classes in the class hierarchy. The technique of indexing attributes for a class hierarchy is called *class hierarchy indexing*, while the conventional approach of indexing attributes per class may be called *single-class indexing*. Experimental studies at MCC show that accesses based on the hierarchy index offer better performance if the level of nesting exceeds two.

The term *nested-attribute indexing* is used to refer to indexing on a class-composition hierarchy. In a nested-attribute index on a class, the attribute indexed is an indirect, nested attribute of the class rather than an attribute of the indexed class. Nested-attribute indexing makes it possible to evaluate a type of complex query by traversing a single index. The type of query for which nested-attribute indexing is ideally suited is one which contains a predicate on a deeply nested attribute of the indexed class.

Maier and others, in the course of theoretical research for the **GEMSTONE** database system [37], have identified some of the issues involved in indexing as follows:

- **Structure Versus Behavior.** This is the issue of whether the indexes should be based on the structure (the attributes) of objects, or the behavior (the responses to messages). Indexing based on the message notation, requires knowledge of the execution model as the system must know which structural changes in an object can influence the result of a message, in order to update the appropriate indexes.
- **Index Structure.** This issue raises questions such as: How deep in the internal structure of an object should we index? Should an index be based on identities of key objects or their values?
- **Indexing on Classes versus Collections.** This is the issue as to whether we should index on classes or collections.

More research needs to be undertaken to investigate the properties of various indexing mechanisms for object-oriented database systems. In particular, the performance of various indexing schemes as well as issues related to updating indexes need to be carefully studied.

4 COMMERCIAL AND PROTOTYPE SYSTEMS

Today, there exists many commercial developments and initial prototypes of object-oriented database systems and systems that provide for the management of persistent objects. Most of these products are still evolving as they are experimenting with advancing technologies and theories in object-oriented technology. Some of these include:

1. **ORION** - MCC Corp., Austin, Tx
2. **GEMSTONE** - Servio Logic Corp.,
3. **OZ+** - University of Toronto
4. **IRIS** - Hewlett-Packard Labs., [21] [22]
5. **VBASE** - Ontologic Corp., Billerica, MA
6. **VISION** - Innovative Systems Techniques, Inc., Newton, MA
7. **TRELLIS/OWL** - Digital Equipment Corp.,

We will now describe in some detail the first two products: ORION and GEMSTONE, as they exhibit some of the more advanced features and are considered to be good representative products in the field of object-oriented database systems. The discussion of these prototypes will focus on the data model and architectural issues discussed in the earlier sections as well as some of the specific features of these products.

4.1 The ORION Database System

ORION is an object-oriented database management system being developed by the Advanced Computer Architecture Program at MCC Corporation, Austin, Texas. It is currently being used to support the data management needs of an in-house expert system shell called Proteus.

4.1.1 Data Model Support

In terms of data model features, perhaps the most distinguishing feature of the ORION system is its support for *composite objects* [31]. As mentioned earlier, composite objects allow us to represent the IS-PART-OF relationship and to capture the notion that objects can be part of other objects. ORION exploits the semantics of composite objects to improve the performance of the system in two ways:

1. **As a unit of clustering.** Here the composite object is used as a unit of clustering in the database, so a large collection of related objects may be stored close to each other and retrieved efficiently from the database. This is because there is a high probability that when an application accesses the root object, it will later access some or all of the dependent objects. In ORION, usually all instances of the same class are placed in the same storage segment on disk. However if it is desirable for instances of multiple classes to share the same segment, the user may specify so by

issuing a *Cluster message*. The cluster message specifies the classes whose instances are to share the same segment.

2. **As a unit of locking.** In this case, the composite object is used as a unit of locking, so that the number of locks that must be set may be minimized in retrieving a composite object from the database. For details of this **granularity locking protocol**, the reader is urged to see [25].

4.1.2 The ORION Architecture

An Overview. The ORION architecture consists of four distinct components/subsystems:

1. **Message handler.** The message handler receives all messages sent to the ORION system.
2. **Object subsystem.** The object subsystem provides high-level functions, such as schema evolution, version control, query optimization, and multimedia information management.
3. **Transaction subsystem.** This subsystem provides transaction support such as concurrency control and recovery.
4. **Storage subsystem.** The storage subsystem provides access and storage management schemes for objects on disk.

Schema Evolution. In ORION, the screening approach as opposed to conversion is utilized to support schema evolution. Thus, all instances of a class are not changed to the new definition of the class, rather, the objects are screened as they are presented to an application and the representations of objects are corrected as they are used. Schema evolution is controlled by a set of invariants and associated rules for preserving the invariants.

Concurrency Control and Recovery. The concurrency control schemes in ORION are based on locking. However the locking schemes are extended to provide three types of *hierarchy locking*. The first is the conventional granularity-hierarchy locking where an explicit lock on any given node of the hierarchy implies locking of all of its descendant nodes. Thus, when a write lock is set on a class, all instances of the class are implicitly locked in write mode. A second locking scheme is used by ORION to ensure that while a class and its instances are being accessed, that no definitions of the class's superclasses (and their superclasses) will be modified. The third locking scheme provides locking for composite objects (as mentioned in the previous section on data model support).

ORION provides transaction recovery mechanisms for soft crashes (which leave the contents of the disk intact) and user-initiated transaction aborts. Recovery against hard crashes (where the contents of the disk are destroyed) by archiving is not currently supported. ORION implements a log-based recovery scheme using UNDO logs. Thus the

"before" values of the attributes of an object are recorded in the log. If a transaction aborts, the log is read backward to back out all updates. When a transaction commits, the log is first forced to disk followed by data updates.

Version Control and Change Notification. In order to support data-intensive application domains, ORION provides support for versions. There exists two types of versions distinguishable by the types of operations that are allowed on them. A *transient version* is one that can be updated and deleted by the user who created it. Also, a new transient version may be derived from an existing transient version and an existing transient version may be promoted to a working version. A *working version* is one that is considered stable and cannot be updated, but may be deleted by its owner. A transient version can be derived from a working version and a transient version can be promoted to a working version. Because of the performance and storage overhead in supporting versions, an application is required to indicate whether a class is *versionable*.

ORION supports change notification as an option on user-specified attributes of a class. An instance is notified of a change to the value of any attribute that is specified as *notification-sensitive*.

Query Processing. ORION supports both forward and reverse traversal mechanisms for traversing the query graph. Reverse traversal is usually mixed with forward traversal

when Boolean operators are present in the query.

4.1.3 Multimedia Information Support

Another prominent feature of ORION is its support for the capture, storage, and presentation of many types of multimedia information including sound and video. This feature is particularly useful for applications in the area of office information systems. The goal is to provide extensibility by providing the ability to add new types of devices and protocols, as well as flexibility and efficiency.

4.2 The GEMSTONE Database System

GEMSTONE is an object-oriented database product developed and marketed by the Servio Logic Corporation of Alameda, California. Theoretical work for the product was done by David Maier and his group at the Oregon Graduate Center [37]. GEMSTONE provides a back-end object-oriented database environment on a SUN-3 or SUN-4 workstation under UNIX or on a DEC VAX under VMS. Typical user access is provided by a VT100 style terminal or a workstation which can be a IBM/PC or Macintosh running SMALLTALK V, or a Sun Workstation or Tektronics workstation running SMALLTALK/80. Communication is established through TCP/IP, typically with ethernet.

4.2.1 Data Model Support

Gemstone provides forty-four predefined classes. Also provided are sequenced and unsequenced collections. Unsequenced collections include *Bag* and *Set* where, presumably, a *Set* is a *Bag* without duplicate entries. Gemstone also provides a set of more than six hundred methods.

4.2.2 The GEMSTONE Architecture

An Overview. The GEMSTONE architecture consists of two components: GEM and STONE. GEM provides user session and communications session control and compilation of OPAL (the object-oriented programming language) methods. STONE provides secondary storage management, concurrency control and recovery, transaction management, authorization, and support for associative access.

Schema Evolution. GEMSTONE utilizes the conversion/coercion approach to the management of schema changes. Thus, when a class/type definition is changed, all instances of the class are changed/converted to the new definition immediately ensuring that auxiliary definitions (such as a class's methods) agree with the new definition. As in ORION, schema changes are guided by *schema modification invariants* that must be preserved across the modifications. One of the long-term goals of the GEMSTONE

project is to support a flexible mechanism in which either screening or conversion may be used appropriately, and further to explore the possibilities of using hybrid mechanisms that incorporate both approaches.

Concurrency Control and Recovery. GEMSTONE utilizes both pessimistic (where conflicting actions are prevented) and optimistic concurrency control mechanisms (where conflicting operations are discarded). The selection of either the pessimistic or the optimistic approach in providing concurrency control to an object depends on the degree of contention on the object. Objects supported by GEMSTONE vary from simple values of variables to complex design objects as may be found in VLSI databases and design applications.

In the context of database recovery mechanisms, GEMSTONE does not maintain any logs of updates. Thus changes made by committed transactions persist, while changes not yet committed are lost. To provide mechanisms to handle media failures, GEMSTONE provides a structure known as a *repository* which is a unit of replication. A Repository is an OPAL (the GEMSTONE supported programming language) class providing an internal representation for repositories. A Repository instance can respond to a message replicate, thus enabling two or more copies of the repository to be maintained online.

Security and Authorization We now mention briefly the various security and authorization schemes provided by GEMSTONE.

1. **Login Authorization.** There are two ways to login to GemStone- through the OPAL programming environment or through the GemStone C interface. A user ID and password is required in both cases to login.
2. **Name Hiding.** It is difficult for a user to refer to global objects that are not in his/her symbol list.
3. **Procedural Protection.** If a program accesses its objects only through methods, then the use of these objects can be controlled by including user identity checks in their methods.
4. **Privileges.** Privilege mechanisms guard a small but crucial set of methods, which exert life-and-death control over all of the GemStone objects and functions.
5. **Authorization by Segments.** Segments are the unit of ownership and authorization . Every user has at least one segment. A user may grant read or write permission on his/her segment to other users.

4.2.3 Language Support

Language support in GemStone is provided by the OPAL language. The OPAL programming language, based on SmallTalk-80, provides data definition, manipulation, and query formulation facilities. It also offers built in data types, operators, and control structures which are comparable to those provided by PASCAL or C. OPAL can also be used to create custom data types (scheme analogues) called classes, to establish the permissible operations on instances of those types.

The OPAL compiler and interpreter call on the GemStone data management kernel to access objects, so all of the benefits of the kernel's services are available in each of the programs. In other words, once a data object is committed to the GemStone database, it persists from session to session. Subsequent OPAL programs can use this data object until it is deleted. These persistent data objects can also be shared by concurrent users who are running different OPAL programs.

Direct terminal and disk I/O facilities are not available in OPAL. Thus, GemStone accepts and transmits byte streams representing GemStone objects (including OPAL code and results) through communication links between the object server and some number of GemStone interface programs running on the server's host or external workstations.

5 RESEARCH DIRECTIONS

We will now highlight some of the areas that are considered to be active areas for research and development.

5.1 Integrating Object-Oriented Languages and Databases

The object-oriented paradigm is now seen by many researchers as a unifying one in programming languages, databases, and artificial intelligence domains. As such we are seeing the addition of the notions of persistency and sharing to object-oriented programming languages while at the same time we are seeing the extension of database systems with object-oriented ideas. A result of these activities could be a single type system and a database programming language for both persistent and non-persistent data.

While the object-oriented framework shows considerable promise as a unifying one for the design of programming languages and databases, some obstacles still remain in the smooth integration of concepts and ideas. Techniques and theories to achieve this integration are now an active research area. Many of these problems arise from the fact that database and programming language design principles and philosophies have differed considerably. Database design principles primarily focused on requirements to provide persistency and sharing of data. On the other hand, in programming languages the focus has always been on processing rather than on data. For a discussion of the conflicts that

arise from these differing philosophies and principles see the article by Bloom [12].

5.2 Transaction Management

The impetus for research in object-oriented database technology has come mainly from applications in CAD/CAM, Engineering, and Office Information Systems. In these domains, there is a need to provide facilities for collaborative and cooperative work. The classical transaction model for databases go to great lengths to ensure that individual user actions are protected from each other. Thus, the issue of transaction management needs to be re-examined thoroughly within the context of these data intensive application domains where collaboration and cooperation among users is a requirement. As an example, transactions may be very long as a user might edit a CAD design object for weeks before releasing it to the public. Proving concurrency control in such an environment opens up the possibility of investigating alternate notions of correctness other than those based on classical serializability theory. Another issue is that of supporting atomic operations.

5.3 Distributed Object-Oriented Database Systems

Most of current research in object-oriented database systems is focusing on centralized systems. However, the need for increased availability and autonomy, improved response times, and other factors will eventually lead to research and the subsequent emergence of distributed (and replicated) object-oriented database management systems. It is only

reasonable not to expect immediate success in this area as many of the issues in the simpler centralized model are still not well understood. A distributed model would have to address among others the following issues:

- **Naming.** Naming schemes in distributed environments should provide users with the capability to designate objects without any concern over their physical location in the network. However, some users may require explicit control over an object's location such as when the need arises to migrate an object to a specific site. Thus, the naming mechanism should provide location-independence as well as the flexibility to the user to control object location.
- **Access.** One of the key issues here is related to handling remote accesses (accesses to objects that reside on remote nodes). One approach is for the requested information about a remote object(s) to be transmitted across the network. Another approach is to migrate the remote object to the local site and then perform a local access. The choice between actual remote access or local access with migration depends on communication load, object size, and application requests.
- **Protection.** The issue of protection is amplified in distributed environments when objects and resources can migrate. In particular, a resource may migrate to a new site, which must then be responsible for protecting this resource.

- **Storage Management and Garbage Collection.** More reserach needs to be done on optimal strategies to cluster and distribute objects in the network. One would have to consider among other things the nature of transactions, queries, complexity and nesting of object structures, response time demands and so on.
- **Transaction Support.** We need to look more deeply into the challenges and problems in supporting atomic actions and transactions in distributed object models. In particular, commit protocols, concurrency control and recovery mechanisms and other techniques have to be investigated and researched.

5.4 Active Database Systems

Should a database merely be something that stores data passively or should the database system as we know it today evolve into an active and intelligent system that can reason, make decisions, and solve complex problems? The answers to this and related questions raise the possibility of what researchers now consider to be active information systems. Thus, we envision the possibility of databases evolving into versatile and intelligent agents.

Recent literature has also defined the notion of *active objects* [19]. Nierstrasz has discussed *Hybrid* [42], an object-oriented programming language in which the objects are active entities. In many object-oriented systems, an object must receive a message before it can perform any action – we could call these passive objects. In contrast, an active

object is one that can initiate actions asynchronously without receiving a message. Thus we can define an active object as one in which a high degree of autonomous responsibility and control is vested. The active object, can be considered as an independent agent and as a source of knowledge and activity.

5.5 Tools and Development Environments

We can expect research in several areas:

- **Database Design Tools.** The problems of logical and physical database design are more complex in the object-oriented paradigm. Thus there is a need for friendly, efficient, and powerful design aid.
- **User Interfaces.** We anticipate user-interfaces with a lot of "object-oriented features". These interfaces will provide advanced behavioral capabilities by supporting operations on data that are semantically close to the abstract data type equivalents of the data objects. Also, the interfaces will provide *direct manipulation* facilities and advanced graphics capabilities.
- **Advanced Software Development Environments.** These environments will provide the capabilities for prototyping and cost-effective software development of applications with object-oriented approaches.

5.6 Optimization Techniques

Research here will focus on theories and principles for optimizing database operations and performance. Thus, optimization techniques for query processing, indexing, storage management, and concurrency control will all be active areas for research.

References

- [1] M. Ahlsen. An architecture for object-management in OIS. *ACM Transactions on Office Information Systems*, 1984.
- [2] T. Andrews. Combining language and database advances in an object-oriented development environment. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 430-440, October 1987.
- [3] F. Bancilhon, W. Kim, and H. Korth. A model of CAD transactions. *Proc. Intl. Conf. on Very Large Data Bases*, pp. 12-21, August 1985.
- [4] J. Banerjee. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, pp. 3-26, January 1987.
- [5] J. Banerjee. Semantics and implementation of schema evolution in object-oriented databases. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, 1987.

- [6] J. Banerjee, W. Kim, and K.C Kim. Queries in object-oriented databases. *Proc. 4th Intl. Conf. on Data Engineering*, February 1988.
- [7] D. Batory and W. Kim. Modeling concepts for VLSI CAD objects. *ACM Transactions on Database Systems*, 10(3), September 1985.
- [8] D. Beech. Groundwork for an object database model. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, The MIT Press, Cambridge, MA, 1987.
- [9] A. Bjornerstedt and C. Hulten. Version control in an object-oriented architecture. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, page , Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [10] A. Bjornerstedt and B. Stefan. ADVANCE: An object management system. In *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pages 206-221, ACM, 1988.
- [11] A. Black. Object structure in the Emerald system. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 67-77, 1986.

- [12] T. Bloom and S.B. Zdonik. Issues in the design of object-oriented database programming languages. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp.441-451, 1987.
- [13] M. Caplinger. An information system based on distributed objects. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 126-137, 1987.
- [14] H.T Chou and W. Kim. Versions and change notification in an object-oriented database system. *Proc. Design Automation Conference*, June 1988.
- [15] G.P. Copeland and S.N. Khoshafian. Identity and versions for complex objects. *Proc. of the 1986 Intl. Workshop on Object-Oriented Database Systems*, 1986.
- [16] B. Cox. Message/Object Programming: an evolutionary change in programming technology. In G.E. Peterson, editor, *Tutorial: Object-Oriented Computing, Concepts*, pages 150-161, The Computer Society Press of the IEEE, 1987.
- [17] K.R. Dittrich. Object-oriented database systems: the notion and the issues. *Proc. of the 1986 Intl. Workshop on Object-Oriented Database Systems*, 1986.

- [18] J. Duhl and C. Damon. A performance comparison of object and relational databases using the Sun benchmark. *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp.153-163, 1988.
- [19] C.A. Ellis and S.J. Gibbs. Active objects: realities and possibilities. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [20] W. Kim et al. Features of the ORION object-oriented database system. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [21] D. Fisherman. IRIS: an object-oriented database management system. *ACM Transactions on Office Information Systems*, 5(1):pp. 48-69, January 1987.
- [22] D. Fishman. Overview of the IRIS DBMS. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [23] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 8(2), June 1983.

- [24] H. Garcia-Molina and K. Salem. *Sagas*. Technical Report CS-TR-70-87, Princeton University, New Jersey, January 1987.
- [25] J.F. Garza and W.Kim. Transaction management in an object-oriented database management system. *Proc. of the ACM SIGMOD Conference*, 1988.
- [26] J.F. Garza and W.Kim. *Transaction Management in an Object-Oriented Database System*. Technical Report ACA-ST-292-87, Microelectronics and Computer Technology Corporation, Austin, Texas, September 1987.
- [27] M.P. Herlihy. Optimistic concurrency control for abstract data types. *Operating Systems Review*, 21(2), April 1987.
- [28] M.P. Herlihy and W.E. Weihl. Hybrid concurrency control for abstract data types. *Proc. of the ACM symposium on Principles of Database Systems*, 1988.
- [29] M. B. Jones and R.F. Rashid. Mach and Matchmaker: kernal and language support for object-oriented distributed systems. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 67-77, 1986.
- [30] S.N. Khoshafian and N.Setrag. Object Identity. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 406-416, 1986.

- [31] W. Kim. Composite object support in an object-oriented database system. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, October 1987.
- [32] W. Kim. A model of queries for object-oriented databases. *Proc. Intl. Conf. on Very Large Data Bases*, August 1989.
- [33] W. Kim. *Object-Oriented Databases: Definition and Research Directions*. Technical Report, Microelectronics and Computer Technology Corporation, Austin, Texas, 1989.
- [34] W. Kim, K.C Kim, and A. Dale. Indexing techniques for object-oriented databases. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [35] R. King. A database management system based on an object-oriented model. In L. Kerschberg, editor, *Expert Database Systems, Proc. from the First Intl. Workshop*, pages pp. 443-468, Benjamin/Cummings, 1986.
- [36] N.A. Lynch. Multilevel atomicity-a new correctness criterion for database concurrency control. *ACM Transactions on Database Systems*, 8(4), December 1983.

- [37] D. Maier. Development of an object-oriented DBMS. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 472-482, 1986.
- [38] D. Maier. Making database systems fast enough for CAD applications. In W. Kim and F. Lochovsky, editors, *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley Publ. Co., Inc., Reading, MA, 1989.
- [39] T. Merrow and J. Laursen. A pragmatic system for shared objects. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, 103-110, 1987.
- [40] J. Micallef. Encapsulation, reusability, and extensibility in object-oriented programming languages. *Journal of Object-Oriented Programming*, 1(1):pp. 12-36, April 1988.
- [41] M. Morgenstern. The role of constraints in databases, expert systems, and knowledge representation. In L. Kerschberg, editor, *Expert Database Systems: Proc. from the First Intl. workshop*, pages pp. 469-483, Benjamin/Cummings, 1986.
- [42] O.M. Nierstratz. Active objects in Hybrid. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 243-253, 1987.

- [43] P.D O'Brien, D.C. Halbert, and M.F. Kilian. The Trellis programming environment. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 91-102, 1987.
- [44] A. Skarra and S. Zdonik. The management of changing types in an object-oriented database. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp.483-495, 1986.
- [45] A. Skarra and S. Zdonik. Type evolution in an object-oriented database. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages pp. 393-416, The MIT Press, Cambridge, MA, 1987.
- [46] T. Watanabe and A. Yonezawa. Reflections in an object-oriented language. *Proc. 3rd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 306-315, 1988.
- [47] P. Wegner. Dimensions of object-based language design. *Proc. 2nd Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 168-182, 1987.
- [48] W.E. Weihl. Commutativity-based concurrency control for abstract data types. *Proc. of the IEEE 21st Annual Hawaii International Conference on System Sciences*, 1988.

- [49] W.E. Weihl. Data-dependent concurrency control and recovery. *Proc. of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983.
- [50] D. Wiebe. A distributed repository for immutable persistent objects. *Proc. 1st Intl. Conf. on Object-Oriented Programming Systems, Languages and Applications*, pp. 453-465, 1986.
- [51] S.B. Zdonik and P. Wegner. Language and methodology for object-oriented database environments. *Proc. of the IEEE 19th Annual Hawaii International Conference on System Sciences*, 1986.